An Energy-Efficient and Error-Resilient Server Ecosystem Exceeding Conservative Scaling Limits

Konstantinos Tovletoglou
Charalampos Chalios
Georgios Karakonstantis
Lev Mukhanov
Hans Vandierendonck
Dimitrios S. Nikolopoulos
Queen's University Belfast

Panos Koutsovasilis
Manolis Maroudas
Christos Antonopoulos
University of Thessaly

Arnau Prat-Perez
Sparsity

M. Mustafa Rafique
Srikumar Venugopal
IBM Research - Ireland

Andreas Diavastos
Zacharias Hadjilambrou
Panagiota Nikolaou
Yiannakis Sazeides
Pedro Trancoso
University of Cyprus

George Papadimitriou
Manolis Kaliorakis
Athanasios Chatzidimitriou
Dimitris Gizopoulos
University of Athens

ABSTRACT

The explosive growth of Internet-connected devices will result in a flood of generated data, which will increase the demand for network bandwidth as well as compute power to process the generated data. Consequently, there is a need for more energy efficient servers to empower traditional centralized (Cloud) data-centers as well as emerging decentralized data-centers at the Edges of the Internet. In this paper, we present our approach, which aims at developing a new class of micro-servers – the UniServer - that exceed the conservative energy and performance scaling boundaries by introducing novel mechanisms at all layers of the design stack. The main idea lies on the realization of the intrinsic hardware heterogeneity and the development of mechanisms that will automatically expose the unique varying capabilities of each hardware component and allow their operation at new Extended Operating Points (EOP). Low overhead schemes are employed to monitor and predict the hardware behavior and report it to the system software. The system software is responsible for optimizing the system operation in terms of energy or performance, while guaranteeing non-disruptive operation under EOP. To efficiently manage any potential fault that may incur under EOP, we aim at identifying critical/vulnerable software structures and developing low cost techniques for protecting them. This eventually, allows us to enhance the fault tolerance of the overall system software that is representative of any state-of-the-art cloud data-center, since it adopts a virtualization environment as well as popular resource management packages. Our initial experiments indicate that there are significant pessimistic margins in processors and DRAMs, and reveal the invariable impact of potential faults on various structures of the system software.

1. INTRODUCTION

The number of intelligent Internet-connected devices is growing day by day and will soon be in the orders of tens of billions, forming the Internet of Things (IoT). Each of these devices is pushing data to the Internet and this data is expected to reach 24.3 exabytes in 2019 [1]. This rapid data growth will put a lot of pressure on the current Internet infrastructure and centralized data-centers, which are already oversubscribed. Coping with this imminent data flood requires not only enhancement of the processing capabilities of the current servers but also rethinking of the way we communicate and process data across the Internet.

A recently introduced approach that has the potential to ensure the viability and scaling of the Internet in the IoT era is *Edge* computing. The premise of Edge computing is to execute services closer to the data sources. Edge computing can reduce application latency [2], and decrease bandwidth requirements between the end user and the data-center. Realizing such an approach requires the design of new server ecosystems that can be deployed closer to the data sources without the need of any expensive cooling or power infrastructure. This is contingent on designing such ecosystems with substantially improved energy efficiency than the current state-of-the-art without compromising performance, availability, programmability, reliability and security properties of the existing cloud data-centers.

Table 1: Sources of variations and voltage guard-bands

Reasons for guard-bands	Voltage Up-scaling
Voltage droops	~20%
Vmin	~15%
Core-to-core variations	~5%

However, realizing such server ecosystems is extremely challenging due to the stagnant voltage scaling (the most effective power saving knob), and the worsening process variations [3], [4] that nanometer circuits are experiencing. In fact, as transistors are being pushed to the atomic scale, it is becoming very difficult to fabricate circuits with the expected specifications leading to large static and dynamic variations [3]. To cope with the significant hardware variability and to hide it from the upper layers of system and application software, manufacturers adopt pessimistic voltage and frequency guard-bands based on the worst-case scenarios. However, such guard-bands limit the circuits to work less efficiently than they could, essentially constraining the power and performance of all manufactured circuits based on the worst-case parts. As shown in Table 1 the voltage guard-bands currently adopted against a variety of issues are already significant. Such margins are becoming more prominent with the use of more cores per chip, the increased voltage droops [5], reliability issues at low voltages (Vmin) [6], and core-to-core variations. As an indication, recent measurements in ARM processors indicated more than 30% timing and voltage margins in 28nm [4].

Realizing that the power and performance overheads imposed by the current pessimistic design paradigm is unavoidable, in this paper we introduce a radical approach that plans to turn the table around by treating the intrinsic hardware heterogeneity as an opportunity and not as a problem. In particular, in UniServer we put forward the following question: Why allow the worse margins of fabricated chips to artificially constrain the performance and energy of today's systems? The reality is that each manufactured processor and each memory module is inherently different and lies on a distinct performance bin (Figure 1). Based on such observation, the UniServer approach plans to substitute the existing conservative margins with the real

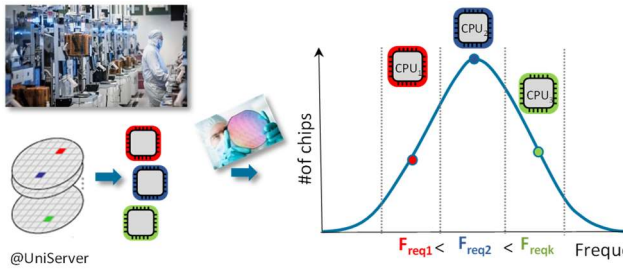

Figure 1: Each manufactured chip is intrinsically different in terms of capabilities

capabilities of each individual core and memory-array. This will enable us to exceed the energy and performance scaling boundaries adopted in servers. In order to achieve this goal, the UniServer project introduces the following technical innovations at all system layers:

i) automatically reveal the possible Extended Operating Points EOP (*i.e.,* voltage, frequency, refresh rate) of each hardware component (*i.e.* cores and memories);
ii) monitor and predict the operating status of the underlying hardware components by introducing new low-level software daemons;
iii) optimize the system operation by adjusting the power/performance/reliability trade-offs based on the enhanced policies and kernel modules;
iv) enable monitoring of the hardware status by all layers of the system software by extending existing interfaces;
v) enhance the fault tolerance of all layers of the system software by providing sufficient protection to critical software structures;
vi) adapt software packages for virtualization (*i.e.* KVM) and resource management (*i.e.* OpenStack) to leverage EOP on next-generation servers;
vii) develop a tool for estimating the Total Cost of Ownership (TCO) gains against other solutions that can be achieved by deploying UniServer in Edge and cloud data-centers; and
viii) analyze security threats in servers operating under the new EOP and provide low cost countermeasures.

With the introduction of the above breakthroughs, UniServer aims at substantially improving the energy efficiency and/or the performance of future servers. UniServer[1] is a project funded by the Horizon 2020 Research Programme of the European Commission that is in its early phase having started on February 2016. The purpose of this paper, is to introduce the proposed approach and present our initial results.

The rest of the paper is organized as follows. Section 2 presents the cross layer UniServer approach. Section 3 discusses the innovation at the hardware and firmware layer. Section 4 discusses the innovation at the System Software. Section 5 compares the proposed approach with existing state of the art. Section 6 discusses the targeted improvements and presents the initial results. Finally, conclusions are drawn in Section 7.

2. THE UNISERVER APPROACH

Figure 1, depicts the different layers of the UniServer ecosystem. The most fundamental idea of the project lies on the hypothesis that each hardware component (*i.e.* core, cache, DRAM) may have intrinsically different capabilities in terms of energy, performance and reliability.

Starting from the low layers we develop techniques that aim at revealing new EOP for each hardware component based on the component's true capabilities. This is achieved by stress-testing the hardware components during a pre-deployment phase under different points using various stress kernels. During deployment, a *HealthLog* daemon is monitoring the health status of the hardware under any used voltage/frequency/refresh rate (V-F-R) point and informs the system software by propagating information vectors about the performance, power, temperature, and any incurred errors. Moreover, another Linux daemon, the *StressLog*, is responsible for periodic offline, on-demand stress testing of the hardware components and for producing an output vector containing the new safe system V-F-R margins that will be suggested to the software (i.e. Hypervisor) for future usage. It also produces log files recording errors (correctable or uncorrectable), system configuration values, sensor readings and performance counters. Using the information provided by the *HealthLog* and *StressLog* the *Predictor* develops probability failure models and tries to predict the hardware behavior under any operating point and eventually helping the system software to decide on the optimum configuration.

[1] UniServer website: http://www.uniserver2020.eu/

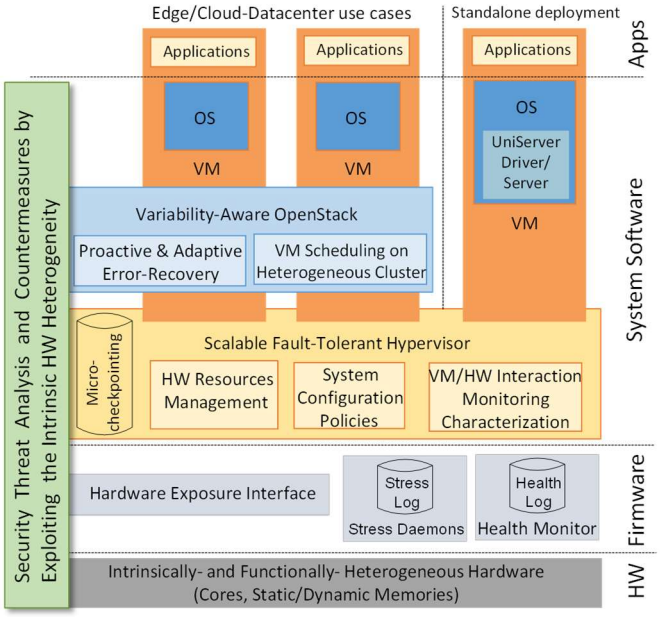

Figure 2: UniServer - Cross-Layer Error-Resilient Ecosystem

UniServer targets a wide range of use cases, ranging from deployments in remote locations close to the end users to deployments in cloud data-centers. To facilitate such diverse use cases, the UniServer platform must be equipped with a complete software stack that can efficiently manage any compute and storage resources by offering easy installation, migration and replication of tasks, either at the node or server-rack level. To this end, state-of-the-art software packages for virtualization (Hypervisor) and resource management (OpenStack) are being adopted. Such packages, apart from managing the virtual machines (VMs) at the node level (Hypervisor) and the resources at a rack/data-center level (OpenStack), they are also being enhanced for optimizing the system operation and the available resources by fine tuning the extended V-F-R points. In particular, the Hypervisor will aim at limiting the effects of the potential faults to higher software layers by reconfiguring the system to operate within safe margins and isolating problematic processing and memory resources that affect the VMs. This is achieved by utilizing the information delivered by the *HealthLog/StressLog/Predictor* daemons and developing a new set of configuration properties. The optimization of operations at the EOP in UniServer is guided by the system requirements of the end-user for each VM, which are typically communicated to the Cloud provider through Service Level Agreements (SLAs). These workload-specific requirements reflect the key metrics of interest based on which OpenStack manages the nodes that constitute any data-centre. Note that in UniServer an additional node reliability metric is added to the traditional metrics of interest, which are node availability, utilization and energy usage. Altogether, these metrics will help in system optimization. The system optimization will be also assisted by developing a tool for estimating the potential TCO gains that can be achieved by various configuration properties of the platform and deployments on Cloud or Edge environments.

The exposure of new EOP, which if not used carefully may result in system failure, entail new security risks. UniServer plans to identify potential security threats (*i.e.,* side channel attacks) that might be caused to micro-servers and develop low cost countermeasures against them. The main chassis of the UniServer is a state-of-the-art 64-bit ARM based Server-on-Chip on which the developed technologies are ported. However, the analysis and developed technologies will not be

tied to a particular platform and special consideration will be given to enable their seamless integration with other servers.

3. Exposing Margins and Monitoring Hardware Behavior

UniServer will use the following technical approach for revealing optimistic margins. Firstly, at the pre-deployment stage, the system goes through a batch of stress-tests to determine the more efficient but safe per-component margins. Secondly, at normal operation in the field, a daemon is constantly recording any possible errors (even if correctable) to fine-tune the margins after deployment. If the number of errors rises above a certain threshold a new stress-test cycle may be triggered to determine new efficient safe margins. This is useful to better adapt to the workloads and also to the aging of the system. Thirdly, during runtime a predictor daemon is running to observe different metrics and advise the Hypervisor on possible execution modes (e.g. high-performance or low-power).

A. Revealing the margins within on-board components

Heterogeneity exists among cores located on the same chip, DRAM and cache memory banks. Each resource may perform better or worse than others but certainly not as any other similar resource on the board. In UniServer we plan to characterize each core and memory bank individually. For example, for each cache memory bank UniServer will reveal the minimum voltage that allows correct operation. This information will be revealed to software and can be exploited towards better energy-efficiency.

B. Stress-test development

First of all, we will stress the underlying cores and memories using diagnostic viruses. We plan to use genetic algorithms for generating these viruses [2], [3]. These viruses will cause maximum voltage noise, power consumption and error rates. The viruses will represent a pathogenic worst case scenario that is unlikely to be encountered in real-life workloads. Safety margins are more pessimistic than these worst-case viruses [1], [2], therefore these stress tests will reveal initial EOP. In addition, real-life workloads will probably allow even more efficient margins since they produce significant less voltage noise, power consumption and error rates compared to stress viruses.

C. HealthLog Daemon

Operating outside the nominal values may introduce hardware errors during the system's lifetime. Thus, there is a need for a runtime mechanism that will monitor the system and report errors occurring during uptime. Such mechanisms already exist for different platforms but important information is missing. Therefore, in UniServer we are extending existing knowledge to create a UniServer-specific monitoring mechanism. We will extend the error reporting capabilities of existing mechanisms with system configuration values, sensor readings and performance counters. We call this mechanism the *HealthLog* monitor that records runtime system metrics in the form of an information vector, stored in a system logfile. The *HealthLog* monitor will also interact and exchange information with higher system layers (e.g. the *Predictor* and the *Hypervisor*). The *HealthLog* monitor will provide two types of services: (a) Event-driven services, where it will collect information based on event occurrences in the system (e.g. errors) and (b) On-demand services, where the monitor will respond to requests from higher layers for specific information.

D. StressLog Daemon

The aim of the UniServer project is to change the nominal V-F-R values, in order to reduce the power consumption of each server in the system. These new values may need to be updated several times over the lifetime of a server due to the aging effects of the machine or unexpected errors observed. Therefore, a mechanism is needed, to produce new nominal values that will still guarantee the safe operations of the server. This mechanism will stress test the machine using predefined applications and compute new safe operating V-F-R margins. We call this mechanism the *StressLo*g monitor.

The *StressLog* monitor will be spawned either periodically during a machines lifetime (e.g. every 2-3 months) or will be triggered by higher system layers in the case of erratic or anomalous machine behavior. The machine being tested will be taken offline and as soon as the monitor receives the input stress target parameters from the higher system layers, it will initiate the stress test scenarios. The *StressLog* monitor will also include a workload suite, consisting of different benchmarks and kernels that either represent real-life applications or are hand-coded to stress specific components of the system. During a stress test, the *HealthLog* monitor will execute in parallel to record system events (errors, system values, sensors and performance counters). The *StressLog* monitor will take the output of the *HealthLog* and will wrap the needed information (defined in the stress target parameters) into a vector to be passed to the higher system layers.

E. Predictor

In order to advise the system regarding the best V-F-R mode depending on the current workload and runtime characteristics of the system, we will develop a machine-learning predictor that interacts with the *HealthLog* and *StressLog* monitors to provide advice to the Hypervisor for choosing the desired operation mode.

4. Managing Operation at Extended Margins at System Software

A. Virtualization

One of the major breakthroughs in the UniServer ecosystem is the ability to explore and allow operation when possible at EOP. In fact, such points may dynamically change depending on the workload, variations of environmental conditions, chip aging etc. and thus the system should be able to decide on the best energy efficient configuration parameters in a fast and reliable way. At the same time, operating so close to the points of failure requires mechanisms to deal with potential, inadvertently introduced faults.

UniServer follows a Hypervisor-based approach based on KVM, to leverage all benefits of virtualization, such as easier deployment, administration, replication and migration, which are necessary for the targeted data-centers at the Edge of the Cloud.

In the context of UniServer, the Hypervisor has additional roles. It is responsible for creating an appropriate execution environment for Virtual Machines (VMs) by manipulating the power/performance/reliability tradeoffs in an educated and safe manner. Specifically, it sets the system at a just-right configuration, which reduces the power footprint of each node by eliminating unnecessary hardware guard-bands, without introducing negative effects on the services running within the VMs. As discussed earlier, the best configuration depends on a number of different parameters, including the characteristics of application software, the capabilities of the specific hardware parts at the specific time and under the specific environmental conditions, as well as the quality of service (QoS) requirements introduced by the cloud management framework (OpenStack).

Despite applying sophisticated configuration policies within the limits specified by the *StressLog*, sporadic errors may still inadvertently occur due to the elimination of guard-bands. The Hypervisor needs to offer VMs a reliable virtual execution environment on top of potentially unreliable hardware. In other words, it needs to transparently mask errors from upper software layers. At the same time, it needs to protect the whole system from catastrophic failures. Being the lowest level of system software, the Hypervisor itself needs to be resilient to errors. Beyond selecting a realistic hardware configuration, the Hypervisor isolates problematic processing and memory resources experiencing high error rates, as reported by the *HealthLog*. This is exactly one of the main aims at the Hypervisor layer and probably less complex than the upper software layers. In particular, the Hypervisor will be enhanced with mechanisms to transparently mask errors from upper software layers, and protect the whole system from catastrophic failures while choosing the right EOP for any given condition/user requirement.

B. Resource Management - OpenStack

The next layer of software is the cloud computing platform. OpenStack [30] makes an ideal candidate for this layer as it is a widely used open source middleware for cloud setups, and it pairs well with the popular enterprise and open source technologies. Our extended version of OpenStack, includes support for monitoring VMs and determining their dynamically changing characteristics and virtual resource utilization at a finer granularity than the existing state-of-the-art. This resource monitoring information will be exploited to design and develop new scheduling policies, as well as to assess the susceptibility of VMs to experience catastrophic errors due to hardware faults. The new scheduling policies, will also focus on incurring minimal overhead and being non-intrusive in real-world scenarios where OpenStack would manage streams of incoming and terminating VMs. Developing such an error-resilient software stack will not only help to avoid system crashes even at EOP but will also help in characterizing and exploring the server operation at aggressive V-F-R scaling points by exploiting the characteristics of real world workloads. Furthermore, by porting the OpenStack on a micro-server will enable resource management capabilities from classical Cloud data centers at the Edge.

5. IMPACT – ENHANCING THE STATE-OF-THE-ART

A. Prior work on DVFS and Variation-Aware Hardware

A wealth of work exists on Dynamic Voltage and Frequency Scaling (DVFS) and turning-off certain parts of the hardware [6] for combatting Dark Silicon and limiting on-chip power consumption. UniServer does not attempt to simply identify the best V-F-R point for a given workload. UniServer attempts to go beyond nominal V-F-R and reveal at runtime new optimistic operating points. Prior suggested attempts to tackle hardware heterogeneity include product binning and utilizing redundancy or using the worst-case for all the parts. With the increasing variability product binning becomes less effective. Many components are being discarded or sold at lower price, thus reducing the yield and revenues [7]. Built-in redundancy (Error Correcting Codes (ECC), extra hardware) may help in maintaining high yield but as the number of faults increases in scaled technologies, the amount of power and resources that are typically wasted is getting large [8].

In addition, pessimistic design margins in voltage and frequency based on the worst-case core or memory cell may reduce the number of faults, however, these do not allow the circuits to operate at the minimum voltage or at the highest frequency that they can, eventually limiting the returns from technology scaling [9]. Instead, UniServer proposes hardware online monitoring and updating the margins accordingly. This way power and silicon is not wasted as with built-in redundancy. Moreover, cost per hardware part may be reduced as parts that previously would have been discarded by binning procedure, will be useful with UniServer approach. UniServer also promises the reduction of data-center acquisition and operating expenses. Related work [1], [10], [11] tries to reduce the margins operating at reduced voltage or higher frequency by occasionally detecting and correcting timing errors that may occur and replaying any faulty instruction. In constrast, the UniServer approach has minimum hardware intrusion and does not require application side modification. UniServer relies on existing hardware detection and correction mechanisms, mechanisms that have become mainstream in high performance and embedded processors.

B. Prior work on Fault Tolerant System Software

A handful of previous works focuses on the implementation of fault tolerant system software. Gu et al. [12] use fault injection to characterize the behavior of the Linux kernel in the presence of faults. In [13] the authors follow a microkernel approach to harden the OS, by moving functionality from the kernel to middleware and then separating system state from the server. Srivastava et al. [14] apply the concept of trust zones between different OS components and control data exchange between components of different levels of trust. Moreover, they restructure OS data structures, so that they can use the standard memory protection mechanisms at the page granularity to control data exchange between trust zones. Finally, the authors in [15] investigate mechanisms to heal the OS in the presence of faults, without rebooting and destroying the state of – potentially unharmed – applications. In the context of Hypervisors, FT-Xen [16] routes all writes to mutable state through a single core which is considered reliable. However, this requires extensive Hypervisor modifications, unless the Hypervisor is inherently non-symmetric. The latter is the case with Xen, which is used as the basis for FT-Xen, but not with KVM. Non-symmetric approaches introduce performance and scalability bottlenecks. At the same time, the correctness is not guaranteed, even on the reliable core, due to the potential propagation of errors from non-reliable cores (through cache coherence or due to non-reliable MCUs). UniServer is based on a symmetric Hypervisor, thus such approaches are not applicable. The UniServer Hypervisor seeks resilience through a careful characterization of the criticality and sensitivity of Hypervisor data structures and code, and educated checking and selective checkpointing mechanisms, driven by this analysis. Other approaches, such as [17] and VMware vLockstep [18] achieve resilience by maintaining coherent replicas of VMs on different physical servers. This approach is not practical neither in Edge computing environments, where replication may not be possible, nor in power- or energy-constrained deployments

Several recent efforts have tried to improve the fault tolerance of a data-center by developing techniques to detect and predict the failures that may occur in a cloud data-center. These techniques [19], [20], [21] generally leverage machine learning or statistical analysis techniques to process the log data generated from the physical or virtual servers to understand the causes of the past failures, and use this information to detect and predict future failures in real time. An unsupervised failure detection and prediction method is proposed in [21] that leverages Bayesian models to improve the reliability and availability of the data-center by detecting the anomaly in the gathered data from the cluster health monitoring tools. Along similar lines, ANCOR [22] proposes a diagnostic system that links resource usage anomalies with the system failures by analyzing the cluster log data. Work presented in [23] uses machine learning approach based on recurrent neural networks for job level and task level failures. A failure prediction method is proposed in [24] for cloud data-centers that uses the pattern of the system log messages to predict a failure by classifying the messages by their similarities in real-time. All these, and similar other, techniques are independent of the cloud middleware and are not integrated with the latest available OpenStack framework. To the best of our knowledge, there is no fault tolerance technique specific to OpenStack framework that detects and predicts system level failures to perform any proactive action to prevent the system failure or to improve the availability of the running application. UniServer's approach is to extend OpenStack framework and have an integrated fault tolerance component, by adapting existing or developing new techniques to efficiently predict the system level failures and proactively migrate the running workloads on the healthy nodes, which is critical to sustain high-availability especially for high value and user-facing workloads

6. SAVINGS PROJECTIONS AND INITIAL RESULTS

As we said, the UniServer project is at its initial phase, focusing on the characterization of the available margins in commercial servers and the identification of the vulnerability of different system software structures. In this section se present our initial characterization results for state-of-the-art cores and DRAMs under different V-F-R points, and different Hypervisor structures. Furthermore, we discuss an initial total cost of ownership analysis for indicating the potential improvements.

A. Characterization of CPUs

We performed an experimental evaluation on two state-of-the-art x86-64 microprocessors, (a low-end Intel Core i5-4200U and a high-end Intel Core i7-3970X) to study: (1) the crash points for each individual core for all the benchmark for voltage offsets below nominal conditions, (2) the core-to-core variation of the crash points among the cores for the same benchmark, and (3) the amount of cache ECC errors

while we reduce the voltage and keep their frequency values unaffected at the highest values. The nominal voltage for the microprocessors under test is 0.844V and 1.365V and the frequency of operation is 2.6GHz and 4.0GHz, for the Intel Core i5-4200U and the Intel Core i7-3970X, respectively. Table 2 presents the findings for our initial experiments using 8 benchmarks (bzip2, mcf, namd, milc, hmmer, h264ref, gobmk, zeusmp) with diverse behaviors from the SPEC CPU2006 benchmark suite [28]:

Table 2: Initial results for two Intel microprocessors

	i5-4200U		i7-3970X	
	min	max	Min	max
crash points below nominal VID	- 10%	- 11.2%	- 8.4%	- 15.4%
core-to-core variation	0%	2.7%	3.7%	8%
number of cache ECC Errors	1	17	-	-

We performed 3 consecutive runs for each benchmark. The crash points present the minimum and maximum offset (as percentage) from the nominal voltage, where the system crashes. Similarly, the core-to-core variation presents the minimum and maximum variability among all available cores for the same benchmark. The min and max values refer to the benchmark that provided the least and the most variability, respectively. The cache ECC errors were exposed only by the low-end microprocessor. On the average for all our experiments, the voltage offset before the crash, where the ECC errors begin to appear is 15mV.

B. *DRAM Characterization*

To quantify the pessimistic margins adopted in the refresh-rate of the DRAMs, we have instrumented a framework for modifying the refresh rate of various 8GB DDR3 DIMMs on a commodity server while running a full-fledged Linux. In our setup, we have separated the main memory into domains (based on the available channels) whose refresh-rate can be set independently. This allowed us to isolate critical kernel code and stack data by placing them on a reliable memory domain (using nominal refresh-rate), and avoid any system crash that may occur under the various relaxed refresh rates that we experimented with. Using random test patterns and various refresh rates, our initial experiments revealed that the refresh rate can be relaxed from 64ms even to 1.5 seconds with no errors introduced to the data, same results found on [32]. Note that the server is in an air-conditioned server room, while the ECC is disabled. In fact, we have observed that even with higher refresh intervals up to 5 seconds (78x higher than the nominal value), the cumulative Bit Error Rate (BER) is in the order of 10^{-9}, which is within the BERs targeted by commercial DRAMs. Note also that classical ECC-SECDED [27] can handle error rates up to 10^{-6}. Relaxing the refresh rate to 1.5 seconds or even to 5seconds can help significantly reduce the refresh power that is responsible for the 9% in today's 2Gb DIMMs and is expected to count for more than 34% of the overall memory power in future 32Gb DIMMs [26].

C. *Error-Resilient System Software*

System Software and especially the Hypervisor of UniServer must be resilient against memory and CPU errors. However, the overhead of resiliency should not outweigh the energy efficiency benefits achieved at EOP. A careful characterization of code and data structures is thus necessary to enable a selective and effective protection strategy. As a first step, we quantitatively evaluated the memory overhead of Hypervisor data structures, with respect to the memory occupied by VMs and applications running on top of it. We measured the Hypervisor memory footprint by repeatedly executing four instances of VMs, each of which accommodates a graph database benchmark (LDBC Social Network Benchmark [29] on top of Sparksee Graph Database). This application stresses the CPU, disk I/O and network. As shown in Figure 3, the Hypervisor footprint (red line) is always less than 7% compared to total utilized memory of the system. Similar observations hold for other applications we experimented with. This dictates placing the whole Hypervisor in a reliable-memory (operated

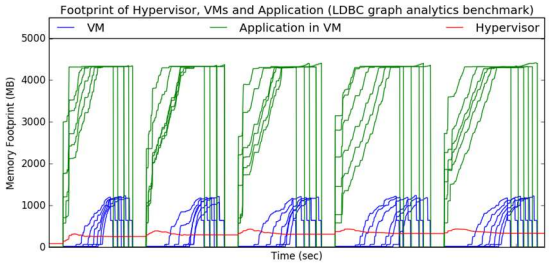

Figure 3: Memory footprint of Hypervisor, VMs and Application

at nominal V-F-R) domain can help ensure non-disruptive operation with low cost.

The Hypervisor can be affected by CPU errors as well. In order to characterize the sensitivity and significance of Hypervisor internal data structures and code, we have applied fault injection using QEMU [33]. More specifically, for each statically allocated object of the Hypervisor (total 16820 objects), we introduced, in independent executions (total 5 executions), Silent Data Corruptions (SDCs). Afterwards, for each execution we checked whether the data corruption resulted to a non-responsive Hypervisor, and marked this object accordingly as crucial or non-crucial for the Hypervisor state. In addition, we experimented both with and without VMs running on top of the victim Hypervisor. Figure 4 depicts the results. It can be observed that the same fault injection rate lead to an order of magnitude more Hypervisor crashes in the presence of active VMs, compared with an unloaded system. At the same time, however, it is clear that there is a clustering in the criticality and sensitivity of data structures and kernel code, according to their functionality. For example, data structures responsible for file system (fs), kernel, network (net) operations are sensitive and should be protected. Interestingly, the sensitive data structures appear to be the same, irrespective of the load on top of the Hypervisor.

D. *Total Cost of Ownership*

Table 3 shows the overall realistic energy efficiency and TCO gains that the UniServer project can achieve in 2019 (estimated project end) over an ARM based server platform. The main sources of the improvements are expected to be (i) technology scaling and leakage reduction due to finfet adoption, (ii) software maturity for ARM based servers, (iii) improved efficiency from running in the Edge, and (iv) operating at EOP using the UniServer approach. By taking in account only the energy efficiency gains we estimate 1.15x TCO improvement. The actual TCO improvement will be even more because of lower chip cost due to higher yield. A tool will be developed in the context of UniServer for end-to-end estimation of the TCO and data-center design exploration. Among other parameters, the TCO tool will consider specific requirements and architecture of both the Cloud and the Edge. It will estimate capital and operational expenses.

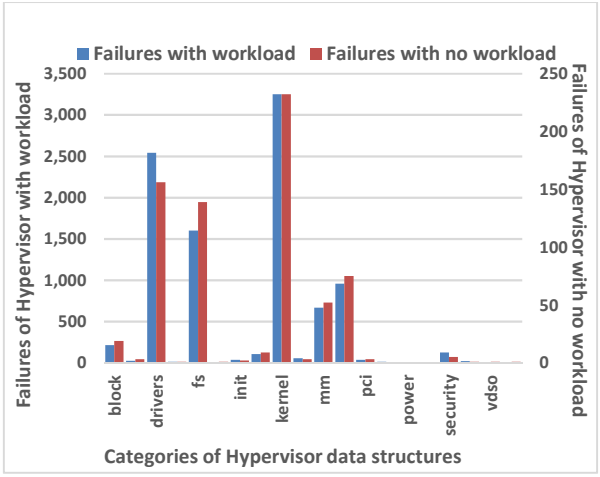

Figure 4: Hypervisor fatal failures in case of errors in different structures

Table 3: Energy efficiency and TCO improvement estimations along with the sources of improvement [31]

EE improvement					TCO
Scaling	Sw maturity	Fog	Margins	Overall	
4	2	3	1.5	36	1.15

By allowing services to run closer to the devices that generate the data it is possible to improve energy efficiency as whole. Specifically, the latency to communicate through the public network to a cloud resource can be leveraged to run a service with much less power or get more work done in the same power envelope. Currently, this latency is in the same order of magnitude with the overall latency targeted by the interactive cloud services (tens to hundreds of milliseconds) [2]. Therefore, a hypothetical IoT service with a target end-to-end latency of 200ms can easily, for a roundtrip to the cloud, expect to spend half of its budget in the network. This leaves a very tight time budget for the actual processing to be done at the data-center. Edge processing has the potential to eliminate most, if not all, of the communication latency and, therefore, can permit to run the service at lower frequency and voltage. For example, operating at 50% of the peak frequency with 30% less voltage translates to running with 50% less energy and 75% less power. Besides addressing the power challenge, the envisioned ecosystem also contributes to assure sustainability, programmability, privacy and security concerns by enabling running services at the Edge. Such services can relieve the public network from the Big Data burden, while ensuring the required quality of service in response and latency sensitive applications. The complete software ecosystem enables seamless administration of cloud and edge data-centers, and reduces the programmability effort that would otherwise be required for porting a service to specialized hardware in the cloud. Finally, the ability of Edge resources to provide a complete service within a home or at the premises of an organization naturally results in improved privacy since the data do not need to be transmitted through the public network and reside in third party data-centers.

7. CONCLUSIONS

In the paper we presented the basic ideas of the UniServer project which attempts to reduce hardware safety margins by utilizing representative stress cases, constant hardware monitoring and predictive mechanisms. The complete system stack approach includes a modified error-resilient Hypervisor and a cloud resource management software. Our initial results indicate the possible margins existing in the state of the art CPUs and DRAMs, while revealing the few kernel structures that are critical for maintaining non-disruptive system operation. In the next 2 years the project plans to realize the testing, monitoring and prediction daemons along with the described fault tolerant hypervisor and OpenStack on a 64-bit ARM based Server-on-Chip. The developed technologies will be evaluated using smart emerging applications deployed in classical cloud business data-centres as well as in new environments closer to the data sources. By doing so the developed prototype aspires to drive Edge computing and turn the opportunities in the emerging Big Data and IoT markets into real, smarter products.

ACKNOWLEDGEMENTS

The presented research effort has received funding from the European Community's Horizon 2020 programme under grant no. 688540 (UniServer). - http://www.uniserver2020.eu/

REFERENCES

[1] Cisco. Visual networking index: Global mobile data traffic forecast update 2014-2019.

[2] HP. White paper. http : // h30507:www3:hp:com/t5/Cloud -Source-Blog/, 2014.

[3] K. A. Bowman, et al. "A 45 nm resilient microprocessor core for dynamic variation tolerance". IEEE JSSC, 2011.

[4] P. N. Whatmough, et al. "14.6 an all-digital power-delivery monitor for analysis of a 28nm dual-core arm cortex-a57 cluster", ISSCC 2015.

[3] Y. Kim, et al. "AUDIT: Stress testing the automatic way", IEEE MICRO, 2012.

[4] A. Bacha, et al. "Dynamic Reduction of Voltage Margins by Leveraging On-chip ECC in Itanium II Processors", ISCA, 2013.

[5] V. J. Reddi et al. "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling." MICRO, 2010.

[6] H. Esmaeilzadeh et al. "Dark silicon and the end of multicore scaling." ISCA, 2011.

[7] S. Borkar et al., "Parameter variations and impact on circuits and microarchitecture," DAC, 2003.

[8] G. Karakonstantis et al. "Containing the nanometer pandora-box: Cross-layer design techniques for variation aware low power systems," IEEE JETCAS, 2011.

[9] L. Leem et al. "Cross-layer error resilience for robust systems" IEEE ICCAD 2010.

[10] S. Das et al., "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance," JSSCC 2009.

[11] D. M. Bull et al., "A Power-Efficient 32 bit ARM Processor Using Timing-Error Detection and Correction for Transient-Error Tolerance and Adaptation to PVT Variation" IEEE JSSC, 2011

[12] W. Gu, et al. "Characterization of Linux Kernel Behavior under Errors". IEEE DSN, 2003.

[13] T.-Y. Lee et al. "Fault isolation using stateless server model in L4 microkernel." ICCAE, 2010.

[14] A. Srivastava et al. "Efficient protection of kernel data structures via object partitioning." In Proceedings of the 28th annual Computer Security Applications Conference, 2012.

[15] F. M. David et al., "Building a self-healing operating system" IEEE DASC, 2007.

[16] X. Jin, et al. "FTXen: Making Hypervisor resilient to hardware faults on relaxed cores." IEEE HPCA, 2015.

[17] T. Bressoud, et al. "Hypervisor-based fault tolerance." ACM TOCS,, 1996.

[18] VMware (2009), 'VMware vSphere™ 4 Fault Tolerance: Architecture and Performance'

[19] A. Bahga et al., "Analyzing Massive Machine Maintenance Data in a Computing Cloud," IEEE TPDS, 2012.

[20] Daniel Dean, et al. "UBL: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems". ICAC, 2012.

[21] P. Gaikwad et al., "Anomaly detection for scientific workflow applications on networked clouds". HPCS, 2016.

[22] Guan, Qiang, et al. "A Failure Detection and Prediction Mechanism for Enhancing Dependability of Data Centers," Journal of Computer Theory and Engineering, 2012.

[23] E. Chuah et al., "Linking Resource Usage Anomalies with System Failures from Cluster Log Data" SRDS, 2013.

[24] X. Chen, et al. "Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study," ISSREW, Naples, 2014, pp. 341-346.

[25] Y. Watanabe, et al. "Online failure prediction in cloud data-centers by real-time message pattern learning," CloudCom, 2012.

[26] J. Liu et al., "RAIDR: Retention-aware intelligent DRAM refresh," ISCA, 2013.

[27] P. J. Nair et al., "ArchShield: architectural framework for assisting DRAM scaling by tolerating high error rates," SIGARCH Comput. Archit., 2013

[28] J. L. Henning, "SPEC CPU2006 benchmark descriptions," SIGARCH Comput. Archit. September 2006.

[29] O. Erling et al., "The LDBC Social Network Benchmark: Interactive Workload," ACM SIGMOD 2015.

[30] OpenStack, "Open source software for creating private and public clouds," [Online]. Available: https://www.openstack.org/.

[31] D. Hardy et al., "An Analytical Framework for Estimating TCO and Exploring Data Center Design Space," IEEE ISPASS, 2013.

[32] J. Liu et al., "An experimental study of data retention behavior in modern DRAM devices: implications for retention time profiling mechanisms," SIGARCH, 2013.

[33] Fabrice Bellard. "QEMU, a fast and portable dynamic translator". *USENIX* ATEC, 2005.